A Comparative Study of Variational Quantum Classifiers and Classical Supervised Models

Víctor Mauricio Gutiérrez Luque

May 12, 2025

Abstract

Contents

Chapter 1

Introduction

Chapter 2

Background and Preliminaries

2.1 Classical Machine Learning Models

2.1.1 Overview of Supervised Learning

Supervised learning is a core approach in machine learning in which the goal is to infer a mapping from inputs to outputs using a dataset of labeled examples. Each example is represented as a pair $(\mathbf{x}, y)$, where $\mathbf{x} \in \mathbb{R}^n$ is an input vector composed of n features, and y is the corresponding target value. The learning algorithm or mathematical function, the model, attempts to approximate a function $f : \mathbb{R}^n \to \mathcal{Y}$ that best predicts y given $\mathbf{x}$, where $\mathcal{Y}$ denotes the set of all possible target values, also called the output or label space (Murphy, 2012).

Some models in supervised learning are called *parametric*, meaning they assume a fixed form for the function f and learn a set of parameters, typically denoted by a weight vector $\mathbf{w} \in \mathbb{R}^n$ and a bias term $b \in \mathbb{R}$ during training. Others are *non-parametric*, which means their structure or complexity grows with the amount of data, without assuming a fixed number of parameters (Murphy, 2012).

In classification tasks, models often produce an intermediate output $z = \mathbf{w}^\top \mathbf{x} + b$, called the score. This is a real-valued scalar that serves as a bridge between raw input features through a decision or *activation function* to final output labels $\hat{y} \in \mathcal{Y}$. For example, a *sign function* $\hat{y} = \text{sign}(z)$ returns $+1$ or -1 based on the sign of z, while probabilistic models may use sigmoid or softmax functions to produce class probabilities. The geometry and dimensionality of the feature space (i.e., how many and which features are used to describe each $\mathbf{x}$) deeply influence how models learn and generalize.

2.1.2 Linear Classifiers

Linear classifiers are a class of models that separate data points in a feature space using a hyperplane defined by a weight vector $\mathbf{w} \in \mathbb{R}^n$ and a scalar bias term $b \in \mathbb{R}$. The model computes a linear combination $z = \mathbf{w}^\top \mathbf{x} + b$, and the sign of z determines the predicted class. This leads to a decision function of the form $f(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x} + b)$, where $\text{sign}(\cdot)$ returns $+1$ for positive inputs and -1 for negative ones (Rosenblatt, 1958).

The vector $\mathbf{w}$ encodes the relative importance of each feature in the input $\mathbf{x}$, while the bias b shifts the decision boundary. The decision boundary itself is the set of points $\mathbf{x}$ for which $\mathbf{w}^\top \mathbf{x} + b = 0$. Linear classifiers rely on the assumption that classes can be separated by such a boundary in the input space (Rosenblatt, 1958). They form the basis of many more complex models and are computationally simple, making them a foundational concept in machine learning.

2.1.3 Logistic Regression

Logistic regression extends the linear classifier framework by interpreting the output of the linear combination $z = \mathbf{w}^\top \mathbf{x} + b$ probabilistically. Specifically, it applies the *sigmoid function*, $\sigma(z) = \frac{1}{1+e^{-z}}$, to map real-valued inputs to the interval $[0, 1]$. This output is interpreted as the estimated probability that the input belongs to the positive class.

Given this probabilistic interpretation, the model is typically trained using *maximum likelihood estimation*, where the parameters $\mathbf{w}$ and b are chosen to maximize the likelihood of the observed labels under the model's predictions. The decision boundary is still linear in the input space, occurring at the point where the model assigns equal probability to both classes (i.e., where $z = 0$).

2.1.4 Support Vector Machine (SVM)

While linear classifiers separate the classes by finding any line, which makes them sensitive to outliers, Support Vector Machines (SVMs), by construction, are margin-based classifiers that aim to find the hyperplane separating two classes, and not focus on outlier points. This makes them more outlier-robust. Given training examples labeled $y_i \in \{-1, +1\}$, the SVM solves an optimization problem that finds the hyperplane $\mathbf{w}^\top \mathbf{x} + b = 0$ such that the distance between this hyperplane and the closest data points (called *support vectors*) is maximized.

Maximizing the margin improves the model's robustness to variations in the data. The supporting hyperplanes, those defining the margin boundaries, are given by $w^T x + b = \pm 1$. The distance from the decision boundary to each of these hyperplanes is $\frac{1}{\|w\|}$. Therefore, the total margin width, which is the distance between the two margin hyperplanes, is $\frac{2}{\|w\|}$. Maximizing it is equivalent to minimizing $\|\mathbf{w}\|^2$, which leads to the primal optimization formulation. This choice regularizes the model by penalizing large weights, which in turn reduces sensitivity to small fluctuations in input values.

The model predicts labels using the *sign function*: $f(\mathbf{x}) = \text{sign}(\mathbf{w}^\top \mathbf{x} + b)$. In this setting, the sign of the inner product determines on which side of the hyperplane the input lies. If the data are not linearly separable, SVMs can be extended using kernel functions, which allow the model to behave as if the data were mapped into a higher-dimensional space, where a linear separator may exist, without explicitly computing that mapping.

2.1.5 k-Nearest Neighbors (k-NN)

The k-Nearest Neighbors (k-NN) algorithm is a non-parametric model that performs classification or regression based on the local structure of the data. Unlike parametric models, k-NN does not learn an explicit function during training. Instead, it retains all training examples and uses them directly during inference. When a new input $\mathbf{x}$ is presented, the algorithm computes the distances between $\mathbf{x}$ and all points in the training set—commonly using Euclidean distance—and selects the k points that are closest.

For classification, the algorithm assigns the label that is most common among these k neighbors. For regression, it typically outputs the average of their values. The underlying assumption in k-NN is that nearby points in the input space tend to have similar outputs. This notion of *locality* is central to its functioning. However, it also means that the performance of k-NN depends heavily on the geometry of the feature space and on how distances are measured. Since it stores and compares all training data at prediction time, it is sometimes called a *lazy learner*.

2.1.6 Kernel Methods

Kernel methods are a set of techniques that allow linear models to learn non-linear patterns by computing similarities between data points using a *kernel function*. A kernel is a function $K(\mathbf{x}, \mathbf{x}')$ that measures how similar two inputs are, without explicitly mapping them into a higher-dimensional space. Instead, it leverages the mathematical property that inner products

in this high-dimensional space can be computed directly via the kernel, a concept known as the *kernel trick*.

One widely used kernel is the Gaussian kernel:

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

This function assigns high similarity to points that are close in Euclidean space and low similarity to distant points. This similarity measure enables the model to construct complex, non-linear decision boundaries in the input space, while still solving a linear problem in an implicit feature space.

2.1.7 Naive Bayes

Naive Bayes is a family of probabilistic classifiers based on Bayes' theorem. It models the posterior probability of a class y given a feature vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ by assuming *conditional independence* between the features given the class. That is,

$$P(y \mid \mathbf{x}) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y).$$

This strong independence assumption simplifies computation, making the model both tractable and interpretable, even though it may not hold exactly in practice.

Naive Bayes classifiers differ based on how $P(x_i \mid y)$ is modeled. For example, the *Gaussian Naive Bayes* model assumes that continuous features are normally distributed within each class. The model computes the posterior probabilities for each class and assigns the one with the highest value. Despite its simplicity, Naive Bayes provides a probabilistic foundation for predicting an output class label and is especially well suited for problems involving discrete or categorical data.

2.1.8 Gaussian Processes (GPs)

Gaussian Processes (GPs) are a non-parametric, Bayesian approach to modeling functions. Instead of assuming a fixed form for the function $f(\mathbf{x})$, a GP defines a *distribution over functions*, such that any finite set of inputs produces a multivariate Gaussian distribution over the outputs. This distribution is specified by a *mean function* $m(\mathbf{x})$, which often defaults to zero, and a *covariance function* $k(\mathbf{x}, \mathbf{x}')$, also known as a kernel, which encodes the similarity between inputs.

When used for classification, the GP provides a prior over latent function values, which are passed through a squashing function (such as the sigmoid) to obtain class probabilities. Because exact inference in this setting is intractable, approximations such as Laplace approximation or variational inference are used.

The key strength of Gaussian Processes is their ability to provide not only predictions but also *uncertainty estimates*. That is, for each prediction, the model returns a distribution (typically Gaussian) reflecting its confidence. This is particularly useful when it is important to know how reliable a model's prediction is, especially in domains like scientific modeling or risk-sensitive decision-making.

2.1.9 Neural Networks (MLPs)

Multilayer Perceptrons (MLPs) are a class of feedforward neural networks consisting of multiple layers of *artificial neurons*. Each neuron computes a weighted sum of its inputs, applies a non-linear *activation function* such as the rectified linear unit (ReLU), sigmoid, or hyperbolic tangent (tanh), and passes the result to the next layer. An MLP typically consists of an *input layer*, one or more *hidden layers*, and an *output layer*.

Formally, each hidden layer performs a transformation of the form:

$$\mathbf{h}^{(l)} = \sigma(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)})$$

where $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the weights and biases of layer l, and $\sigma(\cdot)$ is the activation function. The model is trained using *backpropagation*, which computes gradients of a loss function with respect to each parameter using the chain rule, followed by *gradient descent* to update the weights.

Small MLPs, typically with one or two hidden layers and a moderate number of units, are capable of learning complex, non-linear relationships while maintaining tractability. Unlike kernel methods or instance-based models, MLPs *learn internal representations* of the data during training, which makes them a flexible and general-purpose modeling approach.

2.2 Quantum Computing and Quantum Machine Learning Basics

2.2.1 Quantum Computing

Classical computers represent information using *bits*, which can take values of either 0 or 1. Quantum computing generalizes this idea using *quantum*

bits, or *qubits*. A qubit is a two-level quantum system that can exist in a linear combination of the classical states $|0\rangle$ and $|1\rangle$. A *Hilbert space* is a vector space equipped with an inner product, which allows for the geometric interpretation of quantum states through lengths and angles. Mathematically, a qubit is described as a vector in a two-dimensional complex Hilbert space $\mathbb{C}^2$, with the standard computational basis $\{|0\rangle, |1\rangle\}$ defined as:

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \qquad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

An arbitrary qubit state can be written as:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}, \qquad \alpha, \beta \in \mathbb{C}, \qquad |\alpha|^2 + |\beta|^2 = 1$$

The complex coefficients α and β are called *probability amplitudes*. The state $|\psi\rangle$ is said to be in a *superposition* of the basis states $|0\rangle$ and $|1\rangle$. This means that, unlike a classical bit which is either 0 or 1, a qubit can exist in a linear combination of both states at once. However, when a qubit is measured, it collapses to one of the basis states. Specifically, the outcome will be $|0\rangle$ with probability $|\alpha|^2$, or $|1\rangle$ with probability $|\beta|^2$.

The *tensor product* is a fundamental operation in quantum mechanics used to describe the joint state of multiple quantum systems. Given two vectors $|\psi\rangle \in \mathbb{C}^m$ and $|\phi\rangle \in \mathbb{C}^n$, their tensor product $|\psi\rangle \otimes |\phi\rangle$ is a vector in $\mathbb{C}^{m\times n}$ that represents the combined state of the two systems.

For example, if

$$|\psi\rangle = \begin{pmatrix} a \\ b \end{pmatrix}, \quad |\phi\rangle = \begin{pmatrix} c \\ d \end{pmatrix},$$

then their tensor product is:

$$|\psi\rangle \otimes |\phi\rangle = \begin{pmatrix} ac \\ ad \\ bc \\ bd \end{pmatrix}.$$

Furthermore, entanglement occurs when the quantum states of two or more qubits become non-separable, such that the state of the entire system cannot be expressed as a tensor product of the individual qubit states. Even when the qubits are spatially separated by arbitrary distances, a measurement performed on one qubit instantaneously determines the outcome probabilities of the others.

Using the tensor product, a register of n qubits is constructed by combining individual qubit states. If all qubits are initialized in the state $|0\rangle$, the full n-qubit register is written as:

$$|0\rangle^{\otimes n} = |0\rangle \otimes |0\rangle \otimes \cdots \otimes |0\rangle.$$

As each individual qubit lives in the Hilbert space $\mathcal{H} = \mathbb{C}^2$, the full n-qubit system lives in the tensor product space $\mathcal{H}^{\otimes n} = \mathcal{H} \otimes \mathcal{H} \otimes \cdots \otimes \mathcal{H} \cong \mathbb{C}^{2^n}$.

Operations on quantum states are implemented using *quantum gates*, which are linear transformations represented by unitary matrices. These gates act on qubits and preserve the norm of the state vector. For instance:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Here, X is the Pauli-X gate (quantum NOT), H is the Hadamard gate (used to create superpositions), and Z is the Pauli-Z gate (which flips phase).

Quantum circuits provide the standard computational model for quantum information processing, analogous to Boolean circuits in classical computing. A quantum circuit consists of a sequence of quantum gates acting on a register of qubits. The structure of a quantum circuit reflects the algorithmic logic of the computation, with qubits initialized in known basis states, processed through a network of gates, and finally measured to extract classical information. The expressive power of quantum circuits derives from their ability to exploit superposition, interference, and entanglement, enabling certain tasks to be performed more efficiently than is possible classically.

Grover's Algorithm

To illustrate the operation of a quantum circuit, consider Grover's algorithm, which provides a quadratic speedup for the problem of searching an unstructured database. Grover's algorithm works as followed: given a function $f : \{0,1\}^n \to \{0,1\}$ that evaluates to $f(x) = 1$ for a unique input $x = x_0$, and $f(x) = 0$ otherwise, the goal is to find x_0. Classically, this requires $\mathcal{O}(2^n)$ evaluations in the worst case, whereas Grover's algorithm achieves this in $\mathcal{O}(\sqrt{2^n})$ steps.

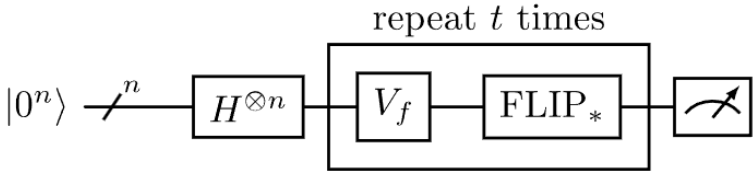

Figure 2.1: The quantum circuit for Grover's algorithm

1. **Initialization**

   We begin with an n-qubit register initialized to the state $|0\rangle^{\otimes n}$. Applying the Hadamard transform $H^{\otimes n}$ yields the uniform superposition:

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{x \in \{0,1\}^n} |x\rangle, \quad \text{where } N = 2^n.$$

   This state assigns equal probability amplitude to every possible input $x \in \{0,1\}^n$.

2. **Oracle Operator**

   The oracle V_f is a unitary operator that flips the sign of the amplitude of the solution x_0. Formally:

$$V_f |x\rangle = \begin{cases} -|x\rangle & \text{if } x = x_0, \\ |x\rangle & \text{otherwise.} \end{cases}$$

   This phase inversion does not reveal x_0, but encodes its identity into the quantum state.

3. **Diffusion Operator or FLIP$_*$**

   The diffusion operator, often referred to as the inversion about the average, amplifies the probability amplitude of the solution. It is defined as:

$$D = 2|\psi\rangle\langle\psi| - I,$$

   where $|\psi\rangle$ is the initial uniform superposition and I is the identity matrix. When applied to a state $|\phi\rangle$, the operator reflects $|\phi\rangle$ about $|\psi\rangle$.

4. **Grover Iteration**

   One Grover iteration consists of applying the oracle followed by the diffusion operator:

$$G = D \cdot V_f.$$

   The state after t iterations is:

$$|\psi^{(t)}\rangle = G^t |\psi\rangle.$$

It can be shown that each application of G rotates the state vector closer toward the solution state $|x_0\rangle$ in a two-dimensional subspace spanned by $|x_0\rangle$ and its orthogonal complement. The rotation angle θ satisfies:

$$\cos(\theta) = \sqrt{\frac{N-1}{N}}, \quad \text{so } \theta \approx \frac{1}{\sqrt{N}} \text{ for large } N.$$

To maximize the probability of measuring x_0, the number of iterations should be approximately:

$$t = \left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor.$$

5. **Measurement and Success Probability**

   After applying t iterations, we measure the quantum state in the computational basis. With high probability (approaching 1 as $N \to \infty$), the outcome will be the desired value x_0. The overall complexity of the algorithm is therefore $\mathcal{O}(\sqrt{N})$, providing a quadratic speedup over classical brute-force search.

2.2.2 Variational Quantum Classifiers (VQCs)

Variational Quantum Classifiers are quantum machine learning models based on *variational quantum algorithms (VQAs)*. These algorithms combine a quantum circuit that depends on a vector of parameters with a classical optimization routine to minimize a cost function.

A VQC receives a classical input vector $\mathbf{x} \in \mathbb{R}^n$. First, the data is encoded into a quantum state using a *quantum feature map*, a unitary transformation $U_\phi(\mathbf{x})$ that acts on the initial state:

$$|\psi_\mathbf{x}\rangle = U_\phi(\mathbf{x})\,|0\rangle^{\otimes n}$$

Here, rotation gates are used to encode classical features into quantum states. For each feature x_j, a corresponding qubit j is rotated by an angle proportional to the feature:

$$\theta_j = c \cdot x_j$$

where c is a scaling constant. Typically, a rotation gate such as $R_y(\theta_j)$ or $R_z(\theta_j)$ is applied. This operation rotates the qubit state on the Bloch sphere, embedding the data into a quantum Hilbert space

Next, a *parameterized quantum circuit* $U(\boldsymbol{\theta})$ is applied to the encoded state. This circuit consists of quantum gates, again typically rotation gates,

with trainable parameters $\boldsymbol{\theta}$ independent of the input data. These are analogous to weights in NNs.:

$$|\psi_{\text{out}}\rangle = U(\boldsymbol{\theta})|\psi_x\rangle$$

After processing, the output state is *measured*. The measurement yields after many runs or shots a probability distribution over classical bitstrings, and this outcome is interpreted to make a prediction. For binary classification, a common approach is to compute the expectation value of a Pauli-Z operator on a target qubit:

$$f(\mathbf{x}) = \langle\psi_{\text{out}}| \, Z \, |\psi_{\text{out}}\rangle$$

The result, which lies between -1 and 1, is post-processed into a binary decision.

The parameters $\boldsymbol{\theta}$ are updated using a classical optimizer that minimizes a cost function, typically comparing predicted outcomes to true labels. This hybrid loop—quantum state preparation, quantum transformation, classical evaluation, and classical optimization—is repeated until convergence (**schuld2018circuit**). VQCs are attractive for implementation on near-term quantum devices because they can operate with shallow circuits and limited quantum resources.

Among the most commonly employed quantum neural network (QNN) architectures in the literature are the dressed QNN and the re-uploading QNN. Both serve as representative baseline models for benchmarking due to their balance between conceptual simplicity and expressive power.

The **dressed variational quantum circuit or d-QNN** is embedded between two trainable classical linear layers. The first linear layer projects the input data into a vector of dimension equal to the number of qubits. These values are then encoded using angle encoding, which maps real-valued inputs to rotation parameters of single-qubit gates. After the trainable circuit ansatz and a local single qubit Pauli-Z expectation value measurements, a second linear layer then maps the measurement results to the target output dimension. This architecture combines classical preprocessing and post-processing with a quantum processing block, which increases flexibility but remains limited in expressivity since each data point is encoded only once (Fellner et al., 2025).

The **re-uploading variational quantum circuit** works by encoding classical data repeatedly at multiple layers of the circuit. Specifically, input features are embedded sequentially using exponential angle encoding, and trainable ansatz layers composed of parameterized gates are inserted between encoding steps. This repeated re-uploading enriches the expressive power

of the model. As with the d-QNN, local Pauli-Z expectation values are measured at the end of the circuit and transformed into the target dimension by a linear output layer (Fellner et al., 2025).

In terms of learning capacity, the d-QNN is structurally similar to a shallow neural network: classical preprocessing and postprocessing layers surround a single quantum circuit with fixed encoding, which limits the model essentially to linear decision boundaries. Its performance is therefore expected to be comparable to that of single-layer perceptrons. By contrast, the ru-QNN repeatedly re-encodes the data at multiple circuit depths, thereby enabling the representation of nonlinear functions. This increased expressivity makes the ru-QNN analogous to a multilayer perceptron (MLP), and its performance can be meaningfully compared to small neural networks.

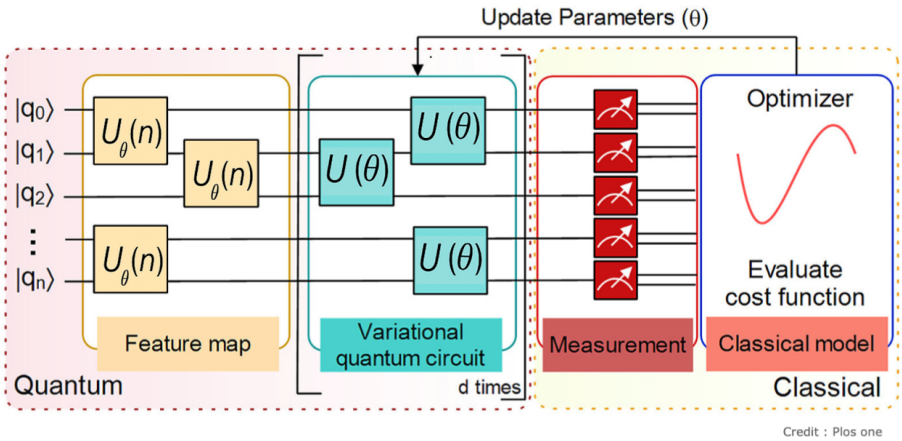

Figure 2.2: Schematic of a Variational Quantum Classifier (VQC), adapted from **mittal2023variational<empty citation>**. The computation begins with n qubits initialized in the state $|0\rangle^{\otimes n}$. The classical input vector $\mathbf{x} \in \mathbb{R}^n$ is embedded into a quantum state via a data-encoding unitary $U_\phi(\mathbf{x})$, known as the quantum feature map. This maps classical data into a high-dimensional Hilbert space. The state is then processed by a parameterized quantum circuit $U(\boldsymbol{\theta})$ with trainable parameters. These variational blocks are often repeated multiple times (denoted by hyperparameter depth d) to increase expressivity. After the final unitary transformation, the qubits are measured. The measurement outcomes are used to compute a cost function, typically defined over expectation values of observables. A classical optimizer then updates the parameters $\boldsymbol{\theta}$ to minimize the cost and this hybrid quantum–classical loop continues until it find this minimal cost convergence.

Chapter 3

Methodology

This study investigates whether Variational Quantum Classifiers (VQCs) can achieve performance comparable to established classical supervised learning models on benchmark datasets. The comparison is meaningful because the selected baselines represent the main paradigms of machine learning: linear (Logistic Regression), kernel-based (Support Vector Machines, Kernel Ridge Regression), probabilistic (Naive Bayes, Gaussian Processes), distance-based (k-Nearest Neighbors), and neural (Multilayer Perceptron). Together, these models provide a comprehensive reference frame for situating VQCs within the landscape of supervised learning.

Past performances and expected. asdasds

3.1 Experimental Setup

3.1.1 Datasets

The experiment are to be conducted on benchmark datasets of varying complexity:

- **Iris, Wine** – Small, well-known datasets with varying class balance and feature correlation.
- **Digits, Fashion-MNIST** – Higher-dimensional datasets to assess model scalability.
- **Blobs, Moons** – 2D separable and non-separable data for visual boundary analysis.

3.1.2 Preprocessing

All datasets were preprocessed to ensure comparability across models and to improve training stability. A key step in this process was *normalization*, which refers to the transformation of input features so that they lie within a standardized range or distribution. This prevents features with larger numeric ranges from dominating the learning process and helps models converge more efficiently.

Two common normalization methods were used depending on the dataset and model:

Min-Max Scaling rescales each feature to a fixed interval, typically $[0, 1]$, using the formula:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Standard Scaling (Z-score Normalization) transforms each feature to have zero mean and unit variance:

$$x' = \frac{x - \mu}{\sigma}$$

where μ and σ are the mean and standard deviation of the feature, respectively.

3.1.3 Dataset Split

After normalization, each dataset was split into *training* and *test* sets to evaluate generalization performance. The training set is used to fit the model parameters, while the test set provides an unbiased estimate of how the model performs on unseen data. To ensure reproducibility, the split was performed using a *fixed random seed*, which guarantees that the same data partitioning is obtained in every run. This makes experimental comparisons reliable and consistent.

3.1.4 Classical Models

Key hyperparameters of each.

3.1.5 Quantum Model: Variational Quantum Classifier (VQC)

- VQC architectures considered.
- Feature maps: ZZFeatureMap, exponential angle encoding.

- Ansatz types: RealAmplitudes, EfficientSU2, TwoLocal.
- Variants: standard VQC, deep VQC, data re-uploading, dressed QNN.
- Qubit counts (after PCA).
- Circuit depths choices.
- Measurement scheme (Pauli-Z expectation values).
- Why these: compatible with Qiskit, reflects common practices in VQC research

3.1.6 Training Procedure

- Classical models: scikit-learn training routines, CV setup.
- Quantum models: optimizers (SPSA, COBYLA, L-BFGS-B), shot counts, number of iterations.
- Backends: Aer simulator (ideal + noisy). Aer simulator is the main simulation framework for running quantum circuits on a classical computer. It has different backends (execution modes),

  Justification: chosen optimizers handle noisy gradients; simulators enable fair and reproducible testing.

3.1.7 Evaluation Metrics

The models were evaluated using:

- Accuracy and F1-Score
- Training time and number of parameters
- Robustness to feature noise
- Performance under small data regimes
- Visualization of decision boundaries

also why these

3.1.8 Experimental Protocol and Tools and Environment

- Train/test split or k-fold cross-validation.

  In addition, *cross-validation* was optionally used during model selection and hyperparameter tuning to improve robustness. In k-fold cross-validation, the training set is divided into k equally sized folds. The model is trained k times, each time using $k-1$ folds for training and the remaining fold for validation. The results are then averaged across folds to obtain a more stable performance estimate. This approach reduces the risk of overfitting to a particular train-test split and provides a better understanding of model performance on limited data.

- Number of repetitions.
- Random seed control.
- Software: Python, scikit-learn, Qiskit.
- Hardware: CPU/GPU specs.
- For transparency, reproducibility, reliability.

Chapter 4

Results and Discussion

This chapter presents the experimental evaluation of Variational Quantum Classifiers (VQCs) in comparison to a range of classical supervised learning models. The discussion is divided into two main parts: a quantitative analysis of model performance and a qualitative discussion of interpretability, complexity, and applicability. The experiments were designed to assess classification accuracy, robustness, and computational feasibility under various settings.

4.1 Quantitative Comparison

This section presents numerical results comparing VQCs and classical models.

- **Accuracy Across Datasets:** Tabulated and graphical comparison of test accuracy.
- **Robustness to Noise and Small Sample Sizes:** Controlled experiments with added noise or reduced training size.
- **Training Time and Resource Use:** Timing results and resource overhead.
- **Visualization of Decision Boundaries**

4.2 Qualitative Analysis

This section provides a broader discussion on the interpretability, feasibility, and practicality of the compared models.

- **Model Strengths and Weaknesses:** Comparative review of performance patterns per model.
- **Interpretability and Complexity:** Transparency of decision functions, number of parameters, ease of tuning.
- **Generalization Capability:** Overfitting tendencies and consistency across datasets.
- **Feasibility in Real-World Scenarios:** Scalability and hardware compatibility for VQC; real-time application potential.

Chapter 5

Conclusion and Future Work

Bibliography

Fellner, T., Kreplin, D., Tovey, S., & Holm, C. (2025). Quantum vs. classical: A comprehensive benchmark study for predicting time series with variational quantum machine learning. *arXiv preprint arXiv:2504.12416*.

Murphy, K. P. (2012). *Machine learning: A probabilistic perspective*. MIT Press.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408. https://doi.org/10.1037/h0042519